

1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2666 2667 2668 2669 2670 2671 2672 2673 2674 2675 2676 2677 2678 2679 2680 2681 2682 2683 2684 2685 2686 2687 2688 2689 2690 2691 2692 2693 2694 2695 2696 2697 2698 2699 2700 2701 2702 2703 2704 2705 2706 2707 2708 2709 2710 2711 2712 2713 2714 2715 2716 2717 2718 2719 2720 2721 2722 2723 2724 2725 2726 2727 2728 2729 2730 2731 2732 2733 2734 2735 2736 2737 2738 2739 2740 2741 2742 2743 2744 2745 2746 2747 2748 2749 2750 2751 2752 2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769 2770 2771 2772 2773 2774 2775 2776 2777 2778 2779 2780 2781 2782 2783 2784 2785 2786 2787 2788 2789 2790 2791 2792 2793 2794 2795 2796 2797 2798 2799 2800 2801 2802 2803 2804 2805 2806 2807 2808 2809 2810 2811 2812 2813 2814 2

Inventor:

Contact: Mikio Ishimaru
(409) 738-0592

METHOD AND SYSTEM FOR PERFORMING FAULT- TOLERANT ONLINE VALIDATION OF SERVICE REQUESTS

TECHNICAL FIELD

The present invention relates generally to computing systems and more particularly to
5 online validation of service requests.

BACKGROUND ART

In the past, a computing system included a number of computers connected by a
network, which allowed the computers to communicate and pass information and/or data
between themselves. The networks ranged from a local system, such as a local area network
10 (LAN), to a very large and expansive network, such as a wide area network (WAN) or the
Internet. The computing system also had various operating systems, storage mediums, and
other processing resources interconnected and accessible via the network through various
software products.

Further, software information service systems and applications were included in the
15 computing system. These systems and applications ranged from commercially available
software applications (e.g. spreadsheets, word processors, databases) to custom developed
software products tailored for specific use within specific computing systems.

Customers or clients (i.e. companies, organizations, groups or individuals) can also be
viewed as part of the computing system. Generally, there are multiple types of clients
20 associated with a distributed computing system including application users, application
developers, and system administrators. Within the distributed computing system, the many
different clients typically require access to a number of different application systems or
processes concurrently and it is necessary to allocate resources and prevent redundant use of
resources.

25 In these types of systems, there is a recurrent problem termed the "online validation
problem" (OVP). In the OVP, a group of processes validate some *request for service* (e.g., if
OVP is used to solve a concurrency control problem, the process is requesting to enter the
critical section). As the result of the validation, only one process accepts the request (i.e.,
following the previous example, only one process enters the critical section), and all the

others reject the request. A request is accepted if it has not been accepted before, and rejected otherwise.

More specifically, a group of processes want to validate some request for service R . Processes may crash during the execution of the validation so a solution must respect two requirements: (a) if a process accepts R , then no other process accepts R , and a process does not accept R more than once; and (b) if no process, among the ones validating R , crashes, then there is a process that eventually accepts R .

Due to process crashes, solving OVP is not simple, as described next.

- (i) Making sure that R is not accepted more than once can be trivially satisfied by a protocol that systematically refuses R . Such solution, however, does not ensure requirement (b). Fulfilling requirements (a) and (b) induce processes to exchange information about the validation of R in such a way that if two (or more) processes validate R , only one accepts it. This is no trivial task because processes have incomplete information about one another.
- (ii) One way to ensure requirement (a) is to have a centralized process that decides whether R should be accepted or rejected. Nevertheless, such a solution has a single point of failure: if the process responsible for the decision crashes, requirement (b) will not be satisfied. A solution to this problem is to detect the crash of processes, but this means that failure detection has to be accurate, something difficult to achieve in most practical cases.
- (iii) Another way is to have transactional database system to solve OVP, however, in order to guarantee that only one process accepts R in case of failures, the transactional database may have to delay the validation until after some processes recover. Transactional-based solutions may delay the result of the validation and make extra assumptions (i.e., no permanent process crash).
- (iv) Solutions relying on perfect failure detection mechanisms can result in unexpected behaviors if the failure detectors make mistakes. Given the unpredictable behavior of commercial network systems, it is hard to tell whether a non-responding process has crashed or is just too busy. It is actually very dangerous to design a system that relies on perfect accuracy of failure detectors. It is difficult to design systems that can rely on unreliable failure detectors because such systems must be

flexible enough to accept wrong failure suspicions, but be strong enough to guarantee that some useful computation is performed.

Solutions to problems similar to OVP largely rely on transactional databases to prevent several processes from accepting the same request R . The key idea is to synchronize transactions (e.g., by means of locking) at some central validation server that only allows one transaction related to R to be active at a time. In such a scheme, the first transaction to lock the database record related to R accepts R , and all the others reject R . Relying on a centralized database may block the system in the event of single crashes, and so, does not fulfill requirement (b) presented in the previous section. A solution to this problem is to use a highly available database, which is expensive.

Database systems supporting asynchronous data replication, such as Tandem Remote Data Facility (RDF) and Microsoft SQL Server, are immediately ruled out because such systems provide weak consistency, and may allow R to be accepted more than once even in executions without crashes. Synchronous data replication systems, such as Oracle Parallel Server (OPS), and Informix Extended Parallel Server (XPS) use clusters with or without shared disks, and do not suffer from the same problem. However, a failover requires log based recovery: if one process takes over for a failed process, it must reconcile its own state with the log of the failed process. Moreover, to use a parallel database system as a highly available transaction processing system, database processes on different machines must access the same disks. This requires special hardware/software, such as high availability clusters.

Several protocols have been proposed to implement “quorum systems”. Quorum systems are a distributed, fault-tolerant synchronization mechanism for replicated databases and objects in general. Although several variants exist, they essentially all detect conflicting requests by means of quorum intersections. Briefly, in order to treat a request r , a server has to gather the approval of a quorum, for example $Q(r)$, of servers. If requests r_1 and r_2 conflict, $Q(r_1)$ and $Q(r_2)$ are such that there is at least one server in any intersection of $Q(r_1)$ and $Q(r_2)$ which detects the conflict, and refuses access to either requests r_1 or r_2 . Quorums are a safety mechanism (i.e., they prevent multiple processes from accepting r), but thus far, no liveness guarantees have been associated with them (i.e., it may happen that even in execution where no process crashes, no process accepts r). Moreover, when used with replicated databases, quorums may lead to distributed deadlocks, which are expensive to resolve.

Mutual exclusion protocols solve the resource allocation problem, which, roughly speaking, requires that at most one process be in the critical section at a time, and if several processes request access to the critical section, one should be granted permission. Given the similarities between OVP and the resource allocation problem, it would be thought that using a mutual exclusion protocol would solve OVP. However, few studies on resource allocation address high availability issues. One study has proposed a modular algorithm for resource allocation in distributed systems that tolerate the failure of some components of the system. However, the study assumes one processor for each resource, and the failure of such processor renders the resource unavailable (although other resources can still be accessed).

Solutions to the OVP have been long sought, but have long eluded those skilled in the art.

DISCLOSURE OF THE INVENTION

The present invention provides a generic protocol with two embodiments that solve the *online validation problem* (or OVP) in distributed computer systems.

In the first embodiment, to validate a request for service R , a process p_i first checks whether R has been validated before (by p_i). Process p_i does this by checking whether R is stored in its local storage. If p_i has not validated R before, p_i broadcasts message $[p_i, R]$ to all processes. Upon delivering a message of the type $[p_k, R]$ for the first time, process p_i (and other processes able to validate R) stores R in its local storage and compares whether the value delivered is the same value it broadcast (i.e., $p_i = p_k$). If so, p_i accepts R ; otherwise p_i rejects R .

In the second embodiment, to validate a request for service R , a process p_i first checks whether R has been validated before (by p_i). Process p_i does this by checking whether R is stored in its local storage. If p_i has not validated R before, p_i sends message $[p_i, R]$ to all processes. Upon receiving a message of the type $[p_k, R]$, every process p_j stores R in its local storage and executes a "Consensus", proposing value $[p_k, R]$ (i.e., p_j executes $propose([p_k, R],)$). By the properties of Consensus, all processes that remain up enough time execute a decide of some proposed value $[p_l, R]$. After process p_i (and other processes able to validate R) delivers $[p_l, R]$, p_i compares whether the value decided is the same value it proposed (i.e., $p_i = p_l$). If so, p_i accepts R ; otherwise p_i rejects R .

The generic protocol implementing OVP solution can be used as a building block to solve abstract problems such as mutual exclusion and quorum allocation, and can also be

used directly to validate electronic tickets or electronic coupons on the Internet. The two embodiments of the generic protocol (a) do not depend on any centralized component, (b) can cope with component failures, (c) make very few assumptions about process and message synchrony, and (d) do not rely on a perfect failure detection mechanism to monitor process failures.

The above and additional advantages of the present invention will become apparent to those skilled in the art from a reading of the following detailed description when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 depicts an example of the execution of a first embodiment of the generic protocol of the present invention; and

FIG. 2 depicts an example of the execution of a second embodiment of the generic protocol of the present invention.

BEST MODE FOR CARRYING OUT THE INVENTION

Referring now to both FIGs. 1 and 2, therein are shown distributed systems 10 and 50. The distributed system 10 has n processes 11, 12, 13, 14, and 15. The distributed system 50 has n processes 51, 52, 53, 54, and 55. A process can be, for example, a server or a UNIX process. The processes do not behave maliciously, and can only fail by crashing. Each process has access to some local storage (not shown), which is not shared with any other process.

The processes are generally connected through reliable channels and, although there may be link failures, any link failure is eventually repaired. In practice, retransmitting messages and tracking duplicates implement the abstraction of reliable channels. The reliable channels are defined by the primitives *send* and *receive*.

The processes can use an "Atomic Broadcast" abstraction or a "Consensus" abstraction, as defined hereinafter. Atomic Broadcast and Consensus are basic abstractions used by many distributed protocols as building blocks. Other than having Atomic Broadcast and Consensus abstractions implemented, the present invention is independent of the relative speeds of the processes and message transmission.

Referring now to FIG. 1, therein is shown an example of the execution of a first embodiment of the generic protocol of the present invention in the distributed system 10, which has processes using the Atomic Broadcast abstraction.

Atomic Broadcast is defined by the primitives *broadcast(message)* and *deliver(message)*, and guarantees that (i) if a process delivers a message m , then all processes eventually deliver m , (ii) if two processes, p_i and p_j , both deliver messages m_1 and m_2 , then p_i delivers m_1 before m_2 if and only if p_j delivers m_1 before m_2 , and (iii) if process p_i broadcasts message m , and remains up enough time, then p_i delivers m .

To validate a request for service R , a process p_i first checks whether R has been validated before (by p_i). Process p_i does this by checking whether R is stored in its local storage. If p_i has not validated R before, p_i broadcasts a message $[p_i, R]$ to all processes. Upon delivering the message of the type $[p_k, R]$ for the first time, process p_i (and other processes able to validate R) stores R in its local storage and compares whether the value delivered is the same value it broadcast (i.e., $p_i = p_k$). If so, p_i accepts R ; otherwise p_i rejects R .

In the distributed system 10 for example, the request for service, a request 20, is provided to the processes 11 through 15. The request 20 is received first by the process 11 and slightly later by the process 14.

In the process 11, the process 11 first checks at validate request 21 whether the request 20 has been validated before by the process 11. The process 11 does this by checking whether the request 20 is stored in its local storage (not shown). When the process 11 has not validated the request 20 before, the process 11 broadcasts 22 a message having information indicating that it originates with the process 11 and is for the request 20. The broadcast 22 is sent to the processes 11 through 15. The request 20 is then stored in the process 11.

In the process 14, the same check is performed at validate request 23 and when the process 14 has not validated the request 20 before, the process 14 broadcasts 24 a message having information indicating that it originates with the process 14 and is for the request 20. The broadcast 24 is also sent to the processes 11 through 15. The request 20 is then stored in the process 14.

In the distributed system 10, the broadcast 24 is delivered 26 to all processes before the broadcast 22 is delivered 28. Upon delivery of the broadcast 24, the processes 11 and 14 store the request 20 in their local storage and then compare whether the value delivered has been validated before.

In the example, since the process 14 has previously validated the request 20, it accepts 30 the request 20.

Since the process 11 delivers 28 the broadcast message 24 before process 11 delivers the broadcast message 22, process 11 rejects the request 20.

5 Referring now to FIG. 2, therein is shown an example of the execution of a second embodiment of the generic protocol of the present invention in the distributed system 50, which has processes using the Consensus abstraction.

10 Consensus is defined by the primitives *propose(value)* and *decide(value)*, and guarantees that (i) every process that remains up enough time decides a value, (ii) no two processes decide a different value, and (iii) if a process decides a value v , then v was proposed by some process.

15 To validate a request for service R , a process p_i first checks whether R has been validated before (by p_i), as done in the first embodiment of the generic protocol. If p_i has not validated R before, p_i sends message $[p_i, R]$ to all processes. Upon receiving a message of the type $[p_k, R]$, every process p_j stores R in its local storage and executes a Consensus, proposing value $[p_k, R]$ (i.e., p_j executes *propose*($[p_k, R]$,)). By the properties of Consensus, all processes that remain up enough time execute a decide of some proposed value $[p_l, R]$. After process p_i (and all other processes able to validate R) delivers $[p_l, R]$, p_i compares whether the value decided is the same value it proposed (i.e., $p_i = p_l$). If so, p_i accepts R ; 20 otherwise p_i rejects R .

In the distributed system 50 for example, the request for service, a request 60, is provided to the processes 51 through 55. The request 60 is received first by the process 51 and slightly later by the process 54.

25 In the process 51, the process 51 first checks at validate request 61 whether the request 60 has been validated before by the process 51. The process 51 does this by checking whether the request 20 is stored in its local storage in the same fashion as for the first embodiment. When the process 51 has not validated the request 60 before, the process 51 sends 62 messages having information indicating that the message originates with the process 51 and is for the request 60. The messages are sent to the processes 51 through 55.

30 In the process 54, the same check is performed at validate request 63 and when the process 54 has not validated the request 60 before, the process 54 sends 64 messages having information indicating that the message originates with the process 54 and is for request 60. The messages are also sent to the processes 51 through 55.

In the distributed system 50, the send 62 from the process 51 is sent before the send 64 from the process 54. The send 62 reaches the processes 51 and 52 first and the send 64 reaches the processes 53 through 55 first.

5 Upon receiving the messages sent, each of the process 51 through 55 stores the request 60 in its local storage and executes a Consensus. The processes 51 and 52 propose 66 a value i representative of the process 51 and the request 60. The processes 53 through 55 propose 68 a value k representative of the process 54 and the request 60. By the properties of Consensus, all processes that remain up enough time execute a decide 70 of i or k . After the process 51 (and all other processes that validate the request 60) delivers i , the process 51
10 compares whether the value decided is the same value it proposed.

In the example, since the value from the process 51 is i , the process 51 accepts 72 the request 60.

Since the value from the process 54 is k , the processes 52 through 55 reject 74 the request 60.

15 While the invention has been described in conjunction with a specific best mode, it is to be understood that many alternatives, modifications, and variations will be apparent to those skilled in the art in light of the foregoing description. Accordingly, it is intended to embrace all such alternatives, modifications, and variations which fall within the spirit and scope of the included claims. All matters hither-to-fore set forth herein or shown in the accompanying drawings are to be interpreted in an illustrative and non-limiting sense.
20